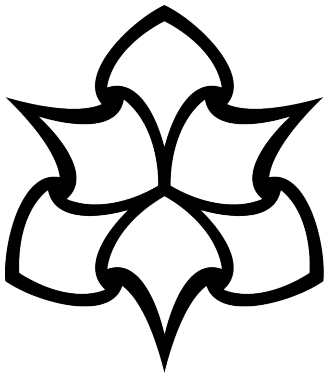

Koo *et al. Journal of Cloud Computing: Advances, Systems and Applications* (2020) 9:50
https://doi.org/10.1186/s13677-020-00195-6

Journal of Cloud Computing: Advances, Systems and Applications

RESEARCH Open Access

IoT-enabled directed acyclic graph in spark cluster

Jahwan Koo[1], Nawab Muhammad Faseeh Qureshi[2*], Isma Farah Siddiqui[3], Asad Abbas[4] and Ali Kashif Bashir[5]

Abstract

Real-time data streaming fetches live sensory segments of the dataset in the heterogeneous distributed computing environment. This process assembles data chunks at a rapid encapsulation rate through a streaming technique that bundles sensor segments into multiple micro-batches and extracts into a repository, respectively. Recently, the acquisition process is enhanced with an additional feature of exchanging IoT devices' dataset comprised of two components: (i) sensory data and (ii) metadata. The body of sensory data includes record information, and the metadata part consists of logs, heterogeneous events, and routing path tables to transmit micro-batch streams into the repository. Real-time acquisition procedure uses the Directed Acyclic Graph (DAG) to extract live query outcomes from in-place micro-batches through MapReduce stages and returns a result set. However, few bottlenecks affect the performance during the execution process, such as (i) homogeneous micro-batches formation only, (ii) complexity of dataset diversification, (iii) heterogeneous data tuples processing, and (iv) linear DAG workflow only. As a result, it produces huge processing latency and the additional cost of extracting event-enabled IoT datasets. Thus, the Spark cluster that processes Resilient Distributed Dataset (RDD) in a fast-pace using Random access memory (RAM) defies expected robustness in processing IoT streams in the distributed computing environment. This paper presents an IoT-enabled Directed Acyclic Graph (I-DAG) technique that labels micro-batches at the stage of building a stream event and arranges stream elements with event labels. In the next step, heterogeneous stream events are processed through the I-DAG workflow, which has non-linear DAG operation for extracting queries' results in a Spark cluster. The performance evaluation shows that I-DAG resolves homogeneous IoT-enabled stream event issues and provides an effective stream event heterogeneous solution for IoT-enabled datasets in spark clusters.

Keywords: Apache spark, Internet of Things (IoT), Directed acyclic graph, MapReduce, Micro-batch stream

Introduction

Real-time streaming empowers an organization to process live data feed generated through an on-line data production system [1]. In the late 90s, an American scientist Peter J. Denning presented a streaming idea to save in-process bits for solving complex calculations much faster than traditional machine processing. This method helps create, process, and observe the data-stream of an instrument and generate a statistical result set [2]. Nowadays, we find several enhanced forms these days such as, live radio, streaming media, HTTP-based adaptive streaming, instant streaming service, HTTP-live streaming, HD streamed video, Full HD (1080p), and streaming 4k content [3]. On the other hand, record-keeping also used a streaming technique to build various data streams management systems such as STREAM, Aurora, TelegraphCQ, and NiagaraCQ [4].

These management systems store data records using a one-time query, long-running query, dataflow query, and query stream; however, it becomes complex for them to manage large-scale dataset queries in a heterogeneous distributed computing environment [5]. Moreover, this

*Correspondence: faseeh@skku.edu
[2]Department of Computer Education, Sungkyunkwan University, Seoul, South Korea
Full list of author information is available at the end of the article

complexity also includes the management of an enormous number of indices in non-tabular datasets that ultimately raises the concept of big data management that could handle large-scale datasets [6]. There are several enterprises out in the market that offers big data management systems such as SQLstream [7], TIBCO [8], IBM [9], Striim [10], and Apache software foundation [11]. Among these, Apache group offers several open-source GPLv3 licensed big data stream engines i.e. Flume [12], Spark [13], Storm [14], NiFi [15], Apex [16], Kafka [17], Samza [18], Flink [19], Beam [20] and Ignite [21], that includes various streaming features as shown in Table 1.

These streaming engines are programmed to handle several forms of data-types, such as structured data, unstructured data, and semi-structured data [22]. These data types are generated through sources that include sensory devices and web-based intelligent portal [23]. Internet of Things (IoT) is a sensory device that consists of an intelligent processor, sensor to detect and store records in its cache storage and an interface to exchange datasets with global networks [24]. This device also generates a continuous flow of data that requires persistent storage to store, and streaming engines categorize its data into three forms, such as unprocessed, processed, and replicas [25]. The unprocessed data is a non-filtered collection that holds an association of tuples with indices only, whereas, the processed data is the extraction of query result onto the unprocessed data. The replica is a block of processed data ready to be exchanged with streaming engines to perform real-time analytics in a distributed computing environment [26], as shown in Fig. 1.

IoT devices also generate several metadata events, i.e., monitoring the temperature of factory devices through smart meters, recording a credit card transaction, and detecting an unwanted object in a surveillance camera [27]. These events are a crucial part of metadata along with logs and routing path information and direct streaming queries to identify data tuples in the repository [28]. By default, streaming through Apache engines involve few steps such as (i) stream sourcing, (ii) stream ingestion, (iii) stream storage, and (iv) stream processing [29]. Stream sourcing represents an IoT device that provides a continuous flow of datasets, and stream ingestion consumes the same sourcing data chunk to queue the tasks inside a streaming engine systematically. The stream storage then formulates a micro-batch, a collection of live data feed having an adequate size s in time t sequentially, and stream processing facilitates the system to execute queries and retrieve a real-time result set [30] as shown in Fig. 2.

The data transformation phase divides micro-batch into four further subtypes, i.e., local generation, file system

Table 1 IoT-based application attribute feature model

FEATURES	Flume	Spark	Storm	NiFi	Apex	Kafka	Samza	Flink	Beam	Ignite
In-Memory	×	√	×	×	√	×	×	×	×	√
Interactive Analytics & Machine Learning	×	√	×	×	√	×	×	√	×	×
Machine-to-Machine	×	×	×	×	×	×	×	×	×	√
High performance transactional processing	×	×	√	×	√	×	×	√	×	√
Data Migration & Binding	√	×	×	√	√	×	×	×	×	×
Fine-grained event processing	×	×	×	×	×	×	×	√	×	×
Low-latency messaging	×	×	×	×	×	√	√	√	×	×
Batch analytics	√	√	√	√	√	√	√	√	√	√
Real-time streaming	√	√	√	√	√	√	√	√	√	√
IoT support	×	√	√	√	×	×	√	×	√	√
Workflow	√	×	×	√	×	×	√	√	√	√
DAG support	×	√	√	√	√	√	√	√	√	√

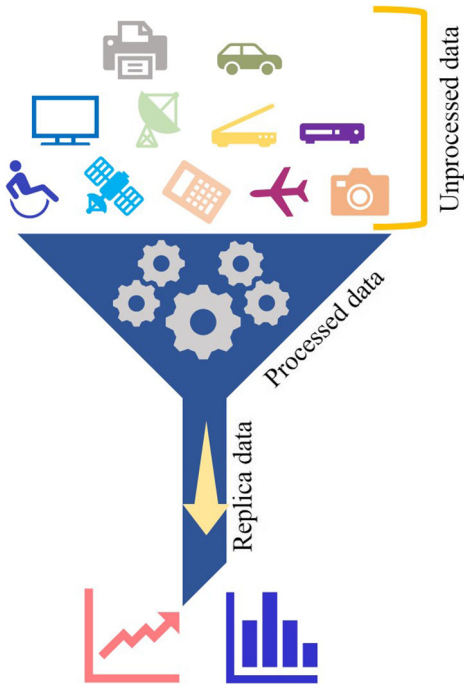

Fig. 1 IoT dataset categorization through Stream Engines

(HDFS) generation, dataset-to-dataset generation, and cache generation [31]. This transformation process is considered relatively *lazy* because of having an abstract extraction of datasets without any real action. Thus, stream processing requires a task route mapper, that could redirect dataset extractions per query into the respective repository. For this, the streaming engine uses a built-in feature of a directed acyclic graph (DAG) that extracts micro-batches to respective column fields without directed cycles [32]. DAG workflow consists of *n* MapReduce stages and transforms micro-batches through a scheduler, which transports dataset through resource allocations using stage functions. By default, a simple DAG consists of $Stage_{0\rightarrow 1}$ stages, whereas, multi-purpose DAG involves $Stage_{0\rightarrow n}$ stages to transform stream into a dataset as shown in Fig. 2a and b.

This workflow facilitates live queries' extraction from a micro-batch; however, it does not recognize the type of IoT data tuples during micro-batch formation. Thus, when processing IoT stream events, it encounters four problems, such as (i) homogeneous micro-batches, (ii) dataset diversification, (iii) heterogeneous data tuples, and (iv) linear DAG workflow issue [33].

This article proposes an IoT-enabled Directed Acyclic Graph (I-DAG) for heterogeneous stream events that minimize the processing discrepancy issue in data transformation. The presented I-DAG enhances workflow operation by reading labeled stream tags in heterogeneous event stream containers and scheduling workflow task processing in a spark cluster. Thus, I-DAG contains additional features of processing IoT tuples and managing the existing DAG properties mentioned below.

The significant contributions of I-DAG are highlighted as:

- A novel event stream tag manager
- A novel parser to filter heterogeneous event streams in the stream engine
- An innovative workflow manager that bypasses the unnecessary tasks queued in stages of MapReduce Operation.
  - $Stage_{0\rightarrow 1}$ I-DAG workflow
  - $Stage_{0\rightarrow n}$ I-DAG workflow

The remaining paper is organized in the following manner. "Motivation" section discusses the benefits

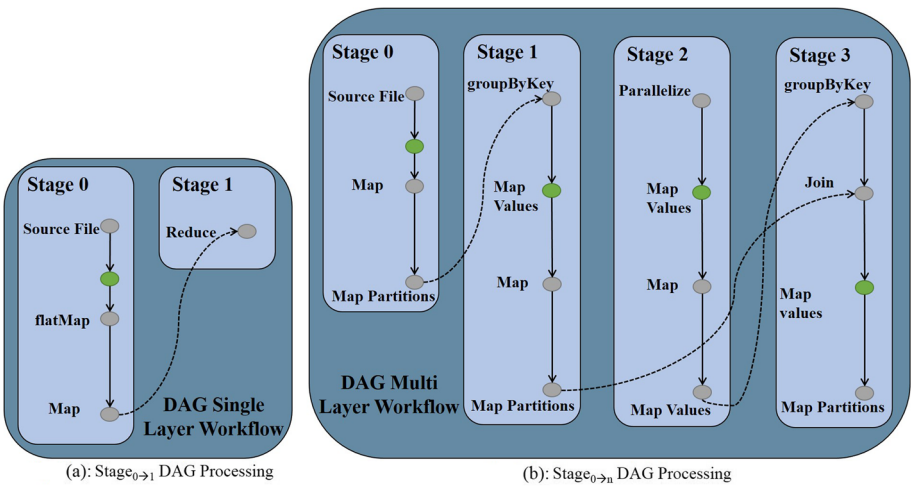

Fig. 2 Default Directed Acyclic Graph (DAG) workflow in Streaming Engine

and complications; "IOT-Enabled directed acrylic graph (I-DAG)" section addresses the motivation; "Performance evaluation" section explains the proposed model I-DAG; "Conclusion" section shows experimental evaluation over the spark cluster. "Declaration" section presents the conclusion with future work.

Motivation

I-DAG is an enhancement in the existing workflow of executing event streams in spark clusters. Let us discuss the benefits and complications of a smart meter use case in a smart grid.

Smart meters cope with on-ground streaming that includes continuous submission of record streams for grid analytics. A smart grid evaluates the functional and procedural performance of distribution end units through that stream. It simultaneously observes the performance of smart meters, i.e., stream accuracy, optimal workload management, and proper functioning of components. A smart grid generates a complicated scenario in bi-directional processing, where a system confirms the accuracy of a stream through the functionality of a source object. Thus, a smart grid cannot verify the accuracy of streaming analytics through a transformed dataset only, but also, it must monitor the error accuracy of smart meters. Therefore, it requires a streaming event analyzer that copes with $Stage_{0\rightarrow n}$ transformations concurrently, and I-DAG provides such features through label-based stream event analytics [34, 35].

Smart meters generate heterogeneous IoT events concurrently through bi-directional streaming that creates asynchronous problems in the smart grid, i.e., outnumbered of metadata than traditional processing and overwhelmed analytical accuracy. Thus, when the I-DAG technique applies, it acquires cache containers to jump few MapReduce tasks that usually a developer skips to include in the programming model [36, 37].

Nowadays, the world is moving towards an unpredictable scale of managing IoT devices and their streaming event analytics. This increment would drastically increase with time, and the demand for resource management would be considered a vital issue that must be managed on a priority basis. At that time, a customized Direct Acyclic Graph for IoT event stream processing would fulfill this demand. This IoT-enabled direct acyclic graph would address future heterogeneous workflow event stream operations in the spark cluster [38].

IOT-Enabled directed acrylic graph (I-DAG)

From a functional perspective, we divide I-DAG into three sub-components:

- Label-based event streaming
- Heterogeneous stream transformation
- IoT-enabled DAG workflow

Label-based event streaming

Let IoT devices events be a sequence of error, backup and information messages with a representation as E_i, B_i and I_i, where each of the message belongs to sensory devices as $Device_i$ in the distributed computing environment as shown in Fig. 3. At each time interval t, streams generated through a function f_i holds an array of event messages $G\,[1..\,(E_i, B_i, I_i)]$ with $G\,[i] = f_i$. Therefore, when a new occurrence of event messages arrive, the function representation changes to $G\,[i++]$ and the individual event message collection at each node could be represented as,

$$G\,[i++] = G\,[(E_i, B_i, I_i) ++] \tag{1}$$

Where, $G\,[i++]$ is a container managing multiple event messages arrival with $x \geq 0$.

In order to approximate the inner function elements of $G\,[i++]$, implicit vectors such as $x\,(E\,[1..n])$, $y\,(B\,[1..n])$ and $z\,(I\,[1..n])$ are added into the stream instruction set with a proportion of $(E_i, x) ++$, $(B_i, y) ++$ and $(I_i, z) ++$ and returns an output approximation as,

$$Event_m = \sum_{i=1}^{n} E_i * B_i * I_i \tag{2}$$

Where, $Event_m > 0$ and represents the container of processed heterogeneous event messages.

Algorithm 1 Labeling the Sensory Device Stream Events $G\,[i++]$

Input Two-stream Deep Hashing [39] onto $G++ : [n] \rightarrow [n]$
Output Result over nodes N_o, N_p and N_q
procedure REDIRECT
  for $i = o$ to m **do**
    $N_o = (-1)^{f(E_i) mod 2} \epsilon \{1, -1\}$
    $N_p = (-1)^{f(B_i) mod 2} \epsilon \{1, -1\}$
    $N_q = (-1)^{f(I_i) mod 2} \epsilon \{1, -1\}$
    $SE_{o,p,q} \leftarrow 0$
    Update:
    $(E_i, x) ++ \leftarrow S_i += x.N_o$
    $(B_i, y) ++ \leftarrow S_j += y.N_p$
    $(I_i, z) ++ \leftarrow S_k += z.N_q$
    Produce
    return $SE_{o,p,q}$
  end for
  $T_1 = \left(SE_{o,p,q}\right)\left(\varepsilon^{-2}\right)$
  $T_2 = \left(SE_{o,p,q}\right)\left(ln\delta^{-1}\right)$
  return $T_1 || T_1$.
end procedure

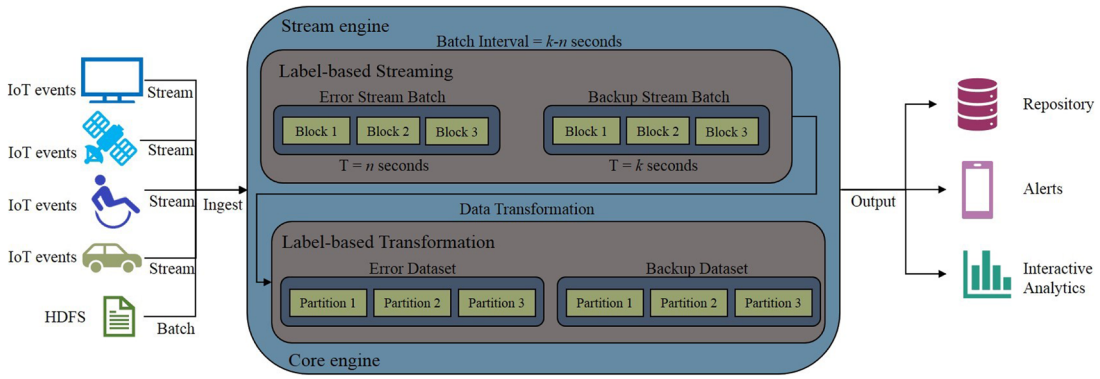

Fig. 3 Label-based Heterogeneous Streaming Workflow

Lemma-1: $SE_{o,p,q} = \sum_{n}^{i=1} \{(e_i \times n_o), (b_i \times n_p), (i_i \times n_q)\}$
The individual data segments of E_i, B_i and I_i arrives at nodes N_o, N_p and N_q through an incremental function $G[i++]$ that assembles segments in formation order. This order summarize stream segments in such a way that $G[i++]$ stores $SE_{o,p,q} \leq 0$.

Lemma-2: $E[s] = PP(e_i, b_i, i_i)$
Since, $SE_{o,p,q} = \sum_{n}^{i=1} \{(E_i \times N_o), (B_i \times N_p), (I_i \times N_q)\}$, but, $\sum_{n}^{i=1} \{(E_i \times N_o) \times (B_i \times N_p) \times (I_i \times N_q)\} \neq N_{o,p,q} \times \left(\sum_{n}^{i=1} (E_i, B_i, I_i)\right)$. Therefore, the constraints are residing within the $(E_i, B_i, I_i,)$. Moreover, if $i = j = k$ then $E[N_{o,p,q}] = E[1] = 1$ and if $o \neq p \neq q$, then $E[N_{o,p,q}]$ are independent and could be retrieved as, $E[N_{o,p,q}] = \frac{1}{2}1 + \frac{1}{2}(-1)$. After that, the linearity of expectation could be represented as,

$$E[SE_{o,p,q}] = E\left[\left(\sum_{i=1}^{n} (E_i, B_i, I_i)\right)\right]\left(\sum_{i=1}^{n} (N_o, N_p, N_q)\right) \tag{3}$$

$$= E\left[\sum_{o,p,q}^{n} (N_o, N_p, N_q)(E_i, B_i, I_i)\right]$$

$$= \sum_{o}^{n} (N_o) E[E_i, B_i, I_i] + \sum_{o \neq p}^{n} (N_p) E[E_i, B_i, I_i] + \sum_{o \neq p \neq q}^{n} (N_q) E[E_i, B_i, I_i]$$

Where $E[SE_{o,p,q}]$ manages the heterogeneous events with independent expectation parameters.

Lemma-3: $V[sE_{o,p,q}] \leq 2E[sE_{o,p,q}]^2$
Since,

$$V[SE_{o,p,q}] = E[(SE_{o,p,q})]^2 - E[SE_{o,p,q}]$$

$$= \left(\sum_{o,p}^{n} \ldots N_o N_p\right) \times \left(\sum_{p,q}^{n} \ldots N_p N_q\right)$$

$$= \sum_{o,p,q}^{n} (\ldots N_o N_p N_q) \leq 2\left(\sum_{o}^{n} E_i, B_i, I_i\right) \times \left(\sum_{p}^{n} E_i, B_i, I_i\right) \times \left(\sum_{q}^{n} E_i, B_i, I_i\right)$$

$$= 2E[SE_{o,p,q}]^2$$

Lemma-4: *average* T_1 *and* T_2 *of* $SE_{o,p,q}$
Let A be the output of algorithm-1, so

$$E[S] = PP(E_i, B_i, I_i), V(A) \leq 2E[A]^2$$

and that equals to the,

$$\sigma(A) = \sqrt{V(A)} \leq \sqrt{2}E[A]$$

Therefore, the bound of stream segment could be obtained as,

$$PE[|A - E[A]| > \varepsilon E[A]]$$

Thus,

$$PE[|A - E[A]| > \varepsilon E[A]] \leq PE\left[|A - E[A]| > \sqrt{2}\varepsilon\sigma(A)\right]$$

In order to reduce the variance, we apply Chebyshev inequality [40] to $\sqrt{2}\varepsilon > 1$, we get the output as,

$$E[A_i] = PP(E_i, B_i, I_i), V(A_i) \leq 2E[A_i]^2$$

So if B be the average of $A_i, \ldots, A_{T_1 T_2}$

$$E[B] = PP(E_i, B_i, I_i), V(B) \leq \frac{2E[B]^2}{T_1 T_2}$$

Now, by Chebyshev's inequality, as $T_1 T_2 \geq \frac{16}{\varepsilon^2}$,

we get,

$$PE\left[|B - E[B]| > \varepsilon E[B]\right] \leq \frac{V(B)}{(\varepsilon H[B])^2}$$

$$PE\left[|B - H[B]| > \varepsilon H[B]\right] \leq \frac{2H[B]^2}{\left(T_1 T_2 \varepsilon^2 H[B]^2\right)} \leq \frac{1}{8}$$

At this point, streaming bound δ could be obtained but since a dependence of $\frac{1}{\delta}$ is present, therefore, we apply lower bound inequality Hoeffding [41] on $H[B] = PP(E_i, B_i, I_i)$ and get,

$$PE\left[(1-\varepsilon)H[B] \leq B \leq (1+\varepsilon)H[B]\right] \geq \frac{7}{8}$$

Now execute median function Z of $T_1 T_2$ onto $B, B_1, ..., B_{T_1 T_2}$ and we get,

$$PE\left[|Z - H[B]| \geq \varepsilon H[B]\right] \leq \delta \tag{4}$$

when,

$$T_1 T_2 \geq \frac{32}{9} ln \frac{2}{\delta}$$

The stream approximation could be obtained as,

$$(E_i)_{N_o} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{HN_{i,i}} \tag{5}$$

$$(B_i)_{N_p} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{BN_{o,p}} \tag{6}$$

$$(I_i)_{N_q} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{IN_{i,k}} \tag{7}$$

This stream approximation defines the existence of managing heterogeneous parameters in the I-DAG.

Heterogeneous stream transformation

The distributed stream elements with probability $\alpha(t)$ are sampled at time t with a computing average of,

$$\alpha(t) = \alpha, contant : error \simeq \frac{1}{\sqrt{\alpha \times t}} \to 0$$

and,

$$\alpha(t) \simeq \frac{1}{\varepsilon^2 \times t} : error \simeq \varepsilon, constant\ over\ time$$

In order to perform encapsulation, reservoir sampling is used because it allows adding first k stream elements to the sample having total items $t - th$ with probability $\frac{k}{t}$. Thus, for every t and $i \leq t$, the sample probability is evaluated as,

$$P_{i,t} = PE\left[s_i\ in\ sample\ at\ time\ t\right] = \frac{k}{t} \tag{8}$$

and for $t+1$, the sample probability becomes,

$$P_{t+1,t+1} = PE\left[s_{t+1}\ sampled\right] = \frac{k}{t+1}$$

This is mandatory because of the inter-connected heterogeneous IoT tuples that are to be incorporated with the internal of time. The processing of $t+1$ with $i \leq t$ eventually reduces the role of s_i and returns s_{t+1} as,

$$P_{i,t+1} = \frac{k}{t} \times \left(1 - \frac{k}{t+1} \times \frac{1}{k}\right) \tag{9}$$

$$= \frac{k}{t} \times \left(1 - \frac{1}{t+1}\right)$$

$$= \frac{k}{t} \times \frac{t}{t+1} = \frac{k}{t+1}$$

The frequency table of stream events uses the event arrival probability $P_{i,t+1}$ into Like space saving of count-min sketch to bring an order between transformed heterogeneous stream events as shown in Fig-3. This space saving function provides an approximation f_x' to f_x for every x and consumes memory equals to $O\left(\frac{1}{\Theta}\right)$. Therefore, when a stream vector $G[n]$ is processed with $G[i] \geq 0$ for $\forall i \epsilon t$, it estimates heterogeneous stream G' of G as,

$$G[i] \leq G'[i] \quad \forall i$$

and,

$$G'[i] \leq G[i] + \varepsilon |G|_1 \quad \forall i, with\ probability\ \geq 1 - \delta$$

Where, $|G|_1 = \sum_i G[i]$ and $|G|_1 \ll streamlength$ having $O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{HN_{i,i}}$, $O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{BN_{o,p}}$ and $O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{IN_{i,k}}$ memory with $O\left(ln \frac{n}{\delta}\right)$ update time t.

The heterogeneous events stream $\sum_i G[i]$ consists of d independent hash functions $h_1 ... h_d : [1..n] \to [1..w]$ where, each of the stream element holds memory $g_p(i)$ that uses instruction set $G[i] += (E_i, B_i, I_i)$ having $g_p(i) += (E_i, B_i, I_i)$ for $\forall j \epsilon\ 1..d$ and the frequency table of heterogeneous events stream could be retrieved as,

$$G'[i] = min\left\{g_p(i) | j = 1..d\right\} \tag{10}$$

This declares that the accessibility of the heterogeneous events stream in enlisted in the I-DAG.

Lemma-5: G' [i] ≥ g [i]

The minimum count of heterogeneous events stream $G[i]$ remains ≥ 0 for $\forall\ i$ with a frequency of $update\left(g_p(i)\right)$. The stream element having hash function $I_{o,p,q} = 1$ if $g_p(i) = g_p(k) = 0$ could be retrieved as,

$$H\left[I_{o,p,q}\right] \leq \frac{1}{range\left(g_p\right)} = \frac{1}{w} \tag{11}$$

By definition $A_{o,p} = \sum_k H\left[I_{o,p,q}\right] \times G[k]$, the heterogeneous events stream can be represented as,

$$A_{o,p} = \sum_{k} H\left[I_{o,p,q}\right] \times G\left[k\right] \leq \frac{|G|_1}{w} \tag{12}$$

Now, this stream is well connected and could not be ready independently. Therefore, we apply Markov inequality and pairwise independence as,

$$PE\left[A_{o,p} \geq \varepsilon\,|G|_1\right] \leq \frac{H\left[A_{o,p}\right]}{\varepsilon\,|G|_1} \leq \frac{\left(\frac{|G|_1}{w}\right)}{(\varepsilon\,|G|_1)} \leq \frac{1}{2} \tag{13}$$

if $w = \frac{2}{\varepsilon}$ then,

$$PE\left[G'\left[i\right] \geq G\left[i\right] + \varepsilon\,|G|_1\right]$$

$$= PE\left[\forall\, j\ : G\left[i\right] + A_{o,p} \geq G\left[i\right] + \varepsilon\,|G|_1\right]$$

$$= PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right] \leq \left(\frac{1}{2}\right)^{d} = \delta \tag{14}$$

$$if\ d = log\left(\frac{1}{\delta}\right)$$

for fixed value of i as shown in Figs. 4 and 5. Thus, we observe that the events are synchronized to a central container with independence of accessibility.

I-DAG workflow

The events generated through IoT devices with a sequential order of $PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right]$are scheduled onto the I-DAG that consists of an identifier $Locator_{I-DAG}$ which reads events labels $(E_i)_{N_o} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{HN_{i,i}}$, $(B_i)_{N_p} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{BN_{o,p}}$ and $(I_i)_{N_q} = O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{IN_{i,k}}$ in the source file and shuffle the pointer between n stages as shown in Fig. 6.

In order to perform stage predictor evaluation, the workflow targets $PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right]$: $stage\,(n) \rightarrow stage\,(n+1)$ with $Locator_{I-DAG}$: $stage\,(n) \rightarrow stage\,(n+1)$ keeping the error under loss function ϑ : $stage\,(n+1) \times stage\,(n+1) \rightarrow \mathbb{R}$. The predictor error can be obtained as,

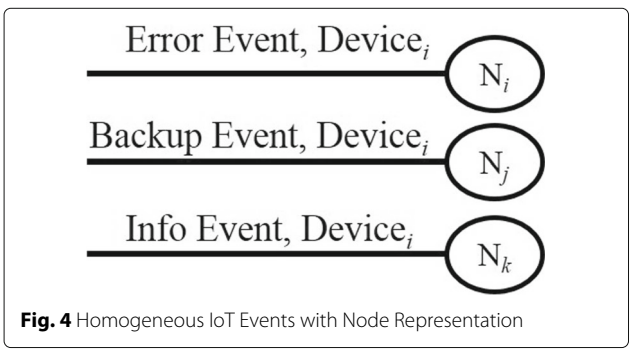

Fig. 4 Homogeneous IoT Events with Node Representation

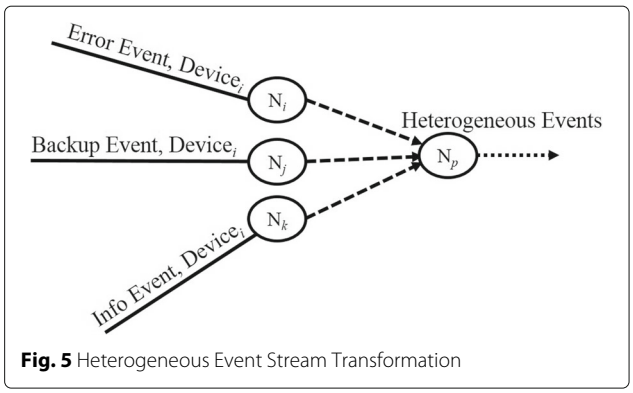

Fig. 5 Heterogeneous Event Stream Transformation

$$H_{stage(n)}\left[\vartheta\left(P\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right]\right.\right.$$
$$\left.\left.\left(stage\,(n)\right), Locator_{I-DAG}\right)\right] \tag{15}$$

This predictor error manages the discrepancies of interconnection in the I-DAG workflow.

The $Locator_{I-DAG}$with an approximated finite heterogeneous event labels can be sampled with $S_{I-DAG} = \left(\left(stage\,(n)_1, stage\,(n+1)_1\right), ..., \left(stage\,(n)_n, stage\,(n+1)_n\right)\right)$ through $\frac{1}{n}\sum_{i=1}^{n} \vartheta\left(P\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right], stage\,(n+1)\right)$. The workflow loss function are categorized into two types: (i) regression and (ii) classification. The regression loss on predictor $Locator_{I-DAG}$ is expressed as,

$$\vartheta\,(a,b) = (a-b)^2 \tag{16}$$

and classification loss on predictor $Locator_{I-DAG}$ is expressed as,

$$\vartheta\,(a,b) = 0\ if\ a = b, 1\ otherwise \tag{17}$$

Thus, I-DAG is ready to facilitate the independent heterogeneous IoT entries with prediction locator.

Performance evaluation

I-DAG technique is incorporated into the Spark cluster, having a virtualized distributed environment, as shown in Table 2.

Table 2 Apache spark cluster

Machine	Specification	No. of VMs	
Intel Xeon E5-2600 v2	8 CPUs, 64 GB memory, 2 TB DISK and 1 TB SSD	3	1 Driver, 2 Workers
Intel Core i7	4 CPUs, 16 GB memory, 1 TB DISK and 500 GB SSD	2	2 Workers
Apache Spark	Spark-2.1.3 (Stable)		
Virtual Machine Monitor	VirtualBox-5.2		

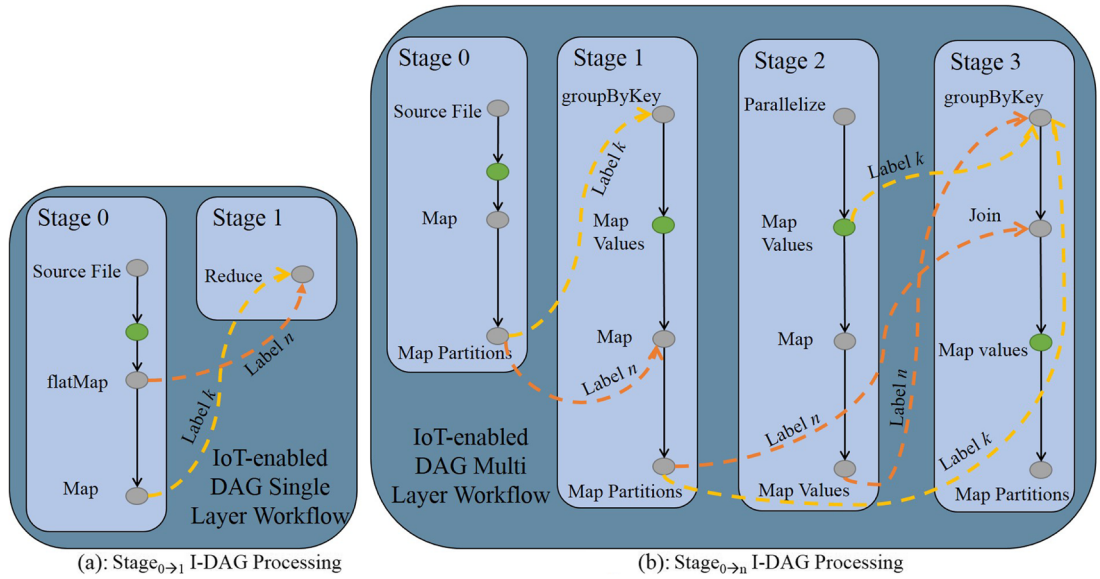

Fig. 6 IoT-enabled Directed Acyclic Graph (I-DAG) workflow in Streaming Engine

Algorithm 2 I-DAG workflow locator $Locator_{I-DAG}$

Input Heterogeneous IoT events $PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right]$
Output Predictor $Locator_{I-DAG}$: $stage\,(n) \rightarrow stage\,(n+1)$
procedure REDIRECT
  for $i = o$ to $PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right]$ **do**
    Read: $^{O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{HN_{i,i}}, O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{BNo_{,p}}, O\left(\frac{1}{\varepsilon^2} ln \frac{2}{\delta}\right)_{IN_{i,k}}}$
    Predict:
    $\frac{1}{n}\sum_{i=1}^{n} \vartheta\left(P\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_1\right], stage\,(n+1)\right)$
    Count
    $k_{ID} = \frac{PE\left[\frac{1}{n}\sum_{i=1}^{n} \vartheta\left(P\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_n\right], stage\,(n)\right) - PE\left[A_{ij} \geq \varepsilon\,|G|_n\right]\right]}{1 - PE\left[\forall\, j\ : A_{o,p} \geq \varepsilon\,|G|_n\right]}$
  end for
  return k_{ID}.
end procedure

Environment

Spark cluster consists of Intel Xeon processor with a core computation capacity of 8 CPU units, 64 GB RAM and persistent storage media of 2 TB Disk and 1 TB SSD. The remaining partial workers consist of the Intel Core i7 processor having 4 Cores, 16 GB RAM, and persistent storage media of 1 TB Disk along with 500 GB SSD. The virtual environment consists of Virtual Box 5.2 installed at five virtual machines, as mentioned in Table 3.

Experiments

The dataset used to evaluate I-DAG belongs to Amazon Web Service (AWS) public datasets repository [42–47]. It contains a collection of 4500 files storing stream data having a total volume of 8.6 GB.

The experiments performed on the AWS dataset consists of (i) Events labeling, (ii) Labeling error factor, (iii) Joining Heterogeneous Streams, (iv) Heterogeneous dataframes, (v) Workflow Endurance and (vii) Cluster performance.

Metrics of evaluation

I-DAG consists of two performance metrics, i.e., (i) Merging of disjoint streams and (ii) Stages bypass. The disjoint stream merging overlaps the individual element and strengthens connectivity between heterogeneous streams. The stages bypass reduces unnecessary consumption of RAM and a decrease in redundant garbage values that appear as a result of regular stage processing.

Results

This section discusses the experimental results generated through the proposed approach I-DAG tasks processing.

Table 3 Virtual machines over spark cluster

Node	CPU	Memory	Storage	Configuration
Driver	6	32 GB	DISK, SSD	Intel Xeon
Worker-1	2	16 GB	DISK, SSD	Intel Xeon
Worker-2	2	16 GB	DISK, SSD	Intel Xeon
Worker-3	2	8 GB	DISK, SSD	Intel Core i7
Worker-4	2	8 GB	DISK, SSD	Intel Core i7

Table 4 Heterogeneous events labeling (seconds)

Streams	Ingest	Queuing	Tagger	Hash	Dispatcher
Error	0.32	0.3	0.15	0.2	0.22
Backup	0.87	0.3	0.14	0.7	0.22
Record	0.09	0.29	0.15	0.1	0.22

Events labeling

IoT devices generate events of errors, backup, and record information in the form of text data that the stream engine receives for micro-batch transformation. Event labels mark the stream elements with an I-DAG tag sequence having a hash function. This tagging creates an impact of trust, and it no longer requires a pair in a prefix or postfix, and the transformation function uses the same hash to bundle stream elements into the core engine. This labeling function consists of several sub-routines such as (i) data ingest, (ii)element queuing, (iii)stream chunk tagger, (iv) hash element, and (v) element dispatcher. The *data_ingest* function fetches an enormous number of individual stream elements from several devices and uses *Heap* memory to enlist the element arrival into stream engine. The *element_queuing* feature then assign the *indices* to respective *Heap* function entries in FCFS (First come First serve) order. The *stream_chunk_tagger* method assigns a label $Stream_{E_i,B_i,I_i}$ to each of the indexed entry and allocates a *hash_element* value for identifying any particular index in the stream and finally the *dispatcher* encapsulates the tags and transform the event streams as shown in Table 4.

The tagged events are recognized by the stream engine much effectively than regular heterogeneous events, as shown in Fig. 7.

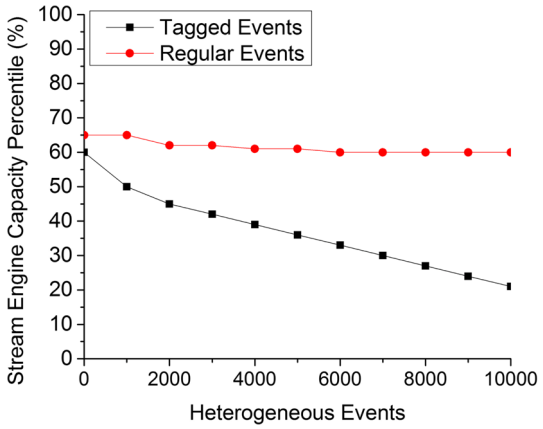

Fig. 7 Heterogeneous Events Processing

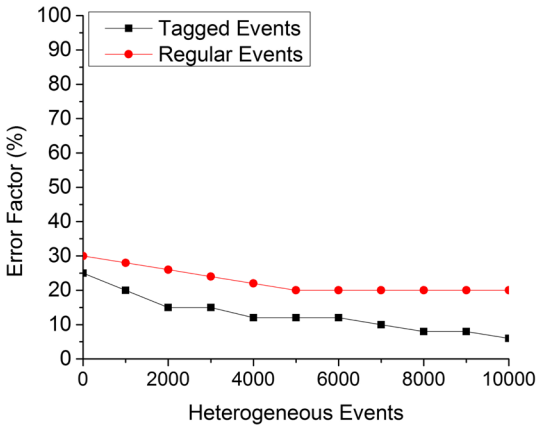

Fig. 8 Errors during Heterogeneous Event Processing

Labeling error factor

The error in an event labeling process appears due to improper placement of tag. It occurs during the application of events labeling relies upon several reasons such as (i) improper ingest, (ii) queue out of bound, (iii) abnormal tagging, (iv) inaccuracy in tag, and (v) partial release of an element. During the tag formation process, a stream element could lead to improper ingestion due to concurrent in-takes at the same time. The queue responsible for managing stream may lead to a buffer overflow problem if the tagging time interval increases than the usual timeline. Also, the stream could be released without having a proper index and hash function due to continuous inaccurate tag application. The errors in label-based events, as well as healthy stream formation, can be observed in Fig. 8.

Heterogeneous streams join

The tagged stream elements require a join operation to combine like events in the stream engine. This requirement is a must because of the live ingestion of heterogeneous stream feed through enormous IoT devices. The functional aspect of a join operation consists of parsing tagged stream elements adjacent to each other so that streaming ingestion must be within the same range of time along with a conjunctive condition that offers to join elements with similar tagging. This conjunction function

Table 5 Heterogeneous stream join through query operator

Events	Parser	Range	Conjunction	Group-by	Aggregation
Error	0.2 sec	0.15 sec	0.03 sec	0.51 sec	0.09 sec
Backup	0.8 sec	0.37 sec	0.06 sec	0.64 sec	0.13 sec
Information	0.12 sec	0.06 sec	0.01 sec	0.26 sec	0.03 sec

Table 6 Heterogeneous stream join through Diff operator

Events	Parser	Range	Conjunction	Group-by	Aggregate	Diff
Error	0.4 sec	0.27 sec	0.05 sec	0.72 sec	0.14 sec	0.6 sec
Backup	0.7 sec	0.41 sec	0.08 sec	0.83 sec	0.25 sec	0.3 sec
Information	0.29 sec	0.08 sec	0.03 sec	0.38 sec	0.08 sec	0.2 sec

correlates element n to $n+1$ through a forward-feed chain in the data transformation environment. The stream element join is executed through syntax $Stream_{E_i} = join\left(parse\left(Tag_n, Tag_{n+1}\right) \rightarrow \left(\left|Tag_n, Tag_{n+1}\right|\right)\right)$ keeping group-by phrase as a priority along with aggregate operators. The heterogeneous streams join of error, backup, and information record events through query operators, as observed through Table 5.

In the same way, the heterogeneous streams join of error, backup, and information record events through diff operator can be observed through Table 6. The comparative effectiveness of the tagged heterogeneous streams joins, as observed through Fig. 9.

Heterogeneous data frames

The label-based stream elements stored in a heterogeneous data frame that comprises a table having data structured properties. This data table assigns a sequence of indices to the stream elements that declare as equal length vectors. The frame categorizes into several subsections, such as (i) header, (ii) data row, and (iii) cell. The header represents the top line of tabular-structure that manages column names only. The data row depicts the stream element having a prefixed index value, and the cell is the stream element member of the row. The data frame supports event labeling transformation through a prior metadata information set of stream elements. Thus, the tagged stream elements are retrieved in a much more efficient manner than traditional stream elements, as shown in Fig. 10.

Workflow endurance

The issues encountered through in-process heterogeneous data streams measures the workflow endurance during stage processing. The IoT-enabled workflow uses data frames to learn about tagged stream elements already enlisted in the data table. Therefore, when a stream joins processes on the source file, the table allows the I-DAG workflow to skip unnecessary steps wherever encountered. This step skipping practice is learned very well through two case studies given as (i) $Stage_{0\rightarrow 1}$ and (ii) $Stage_{0\rightarrow n}$. The $Stage_{0\rightarrow 1}$ consist of two stages having three operations in total that includes *flatMap*, *Map* and *Reduce*. If the label stream element already processed through the *Map* functionality, it can jump the control from *flatMap* to *Reduce* operation. In the case of $Stage_{0\rightarrow n}$, when the compiler parses the source file that consists of a

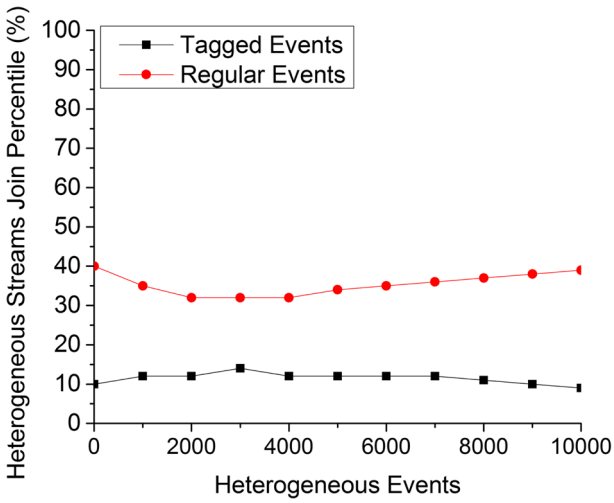

Fig. 9 Heterogeneous Streams Join Computing Percentile

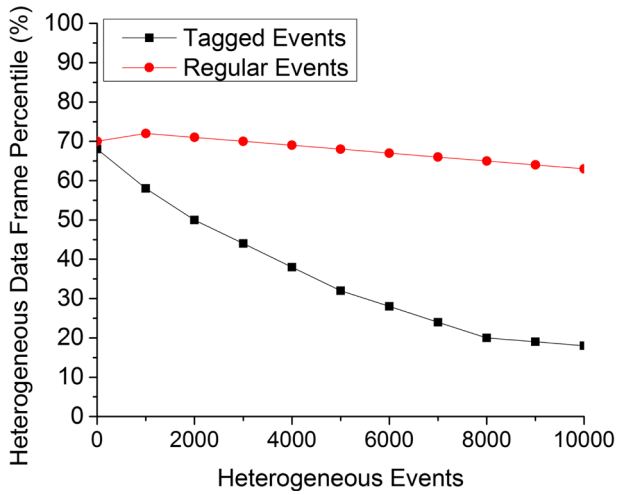

Fig. 10 Heterogeneous Data Frame Computing Percentile

schedule, the control bypasses unscheduled operations in the stages. Thus, it reduces the usage of energy consumption and the computing capacity of a cluster along with skipping functional latency issues. The $Stage_{0\rightarrow 1}$ and $Stage_{0\rightarrow n}$ performance could be observed through Tables 7 and 8.

Cluster performance

The parameters measuring cluster performance comprise stage activity that includes map and reduce task processing and the exchange of i/O operations. i-DAG enables a cluster to perform switching in-between stage tasks depending on the source file's requirement. if the task does not require to produce map values, it bypasses the operation towards the next task, unlike traditional dAG that has to go through each of the individual operation producing i/O latency along with additional operational cost as shown in Fig. 11.

Conclusion

This paper proposes a novel technique that identifies different IoT devices stream events over a graph processing layer in spark cluster. The proposed approach provides a broad analytical perspective of how the stream events are generated, proceeded by their convergence in the heterogeneous form. In the end, the I-DAG workflow processes individual IoT devices' stream events with a cost-effective mechanism. It reduces graph workload along with decreasing the I/O traffic load in the spark cluster.

Table 7 Heterogeneous stream I-DAG workflow $Stage_{0\rightarrow 1}$

Event	Query type	Source file	Flat map	Map	Reduce	Core %	DISK (GB)	I-DAG
Error	Simple	2.3 sec	20 sec	Bypassed	29 sec	21.2%	0.0002%	2-node
Backup	Simple	3.8 sec	28 sec	25 sec	31 sec	22.9%	0.0008%	3-node
Informaton	Simple	1.9 sec	15 sec	Bypassed	21 sec	19.3%	0.0003%	2-node
Error	Compound	18 sec	41 sec	10 sec	49 sec	23.3%	0.0004%	3-node
Backup	Compound	22 sec	39 sec	Bypassed	41 sec	18.71%	0.0003%	2-node
Information	Compound	16 sec	37 sec	Bypassed	39 sec	20.66%	0.0008%	2-node
Error	Range	35 sec	12 sec	19 sec	37 sec	21.97%	0.0008%	3-node
Backup	Range	42 sec	29 sec	Bypassed	43 sec	19.38%	0.0009%	2-node
Information	Range	28 sec	11 sec	22 sec	41 sec	21.26%	0.0007%	3-node
Error	String	17 sec	15 sec	Bypassed	29 sec	22.3%	0.0009%	2-node
Backup	String	16 sec	18 sec	31 sec	38 sec	20.61%	0.0004%	3-node
Information	String	18 sec	13 sec	43 sec	39 sec	19.52%	0.0003%	3-node

Table 8 Heterogeneous stream I-DAG workflow $Stage_{0 \rightarrow n}$

Event	Query	Source	Map	Partition	Value	GroupBy	Join	Equal	Reduce	Core%	DISK-GB	I-DAG
Error	Simple	4.9 sec	25 sec	38 sec	62 sec	Bypassed	22 sec	Yes	63 sec	30.5%	0.0009%	5-node
Backup	Simple	8.3 sec	32 sec	Bypassed	87 sec	58 sec	Bypassed	Yes	58 sec	29.7%	0.009%	4-node
Informaton	Simple	7.6 sec	26 sec	47 sec	Bypassed	43 sec	12 sec	Yes	46 sec	28.64%	0.0006%	4-node
Error	Compound	63 sec	68 sec	Bypassed	49 sec	38 sec	Bypassed	Yes	86 sec	31.75%	0.0009%	4-node
Backup	Compound	48 sec	75 sec	48 sec	Bypassed	59 sec	18 sec	Yes	63 sec	32.41%	0.0009%	5-node
Information	Compound	71 sec	81 sec	52 sec	46 sec	Bypassed	31 sec	Yes	75 sec	29.63%	0.0008%	5-node
Error	Range	85 sec	49 sec	Bypassed	71 sec	51 sec	Bypassed	Yes	67 sec	30.76%	0.0008%	4-node
Backup	Range	76 sec	57 sec	63 sec	Bypassed	78 sec	37 sec	Yes	82 sec	29.66%	0.0009%	5-node
Information	Range	84 sec	64 sec	Bypassed	69 sec	83 sec	Bypassed	Yes	59 sec	31.69%	0.0006%	4-node
Error	String	74 sec	46 sec	59 sec	Bypassed	48 sec	18 sec	Yes	71 sec	29.48%	0.0008%	5-node
Backup	String	69 sec	39 sec	Bypassed	41 sec	88 sec	34 sec	Yes	68 sec	30.53%	0.0007%	5-node
Information	String	82 sec	51 sec	61 sec	Bypassed	49 sec	71 sec	Yes	84 sec	29.83%	0.0007%	5-node

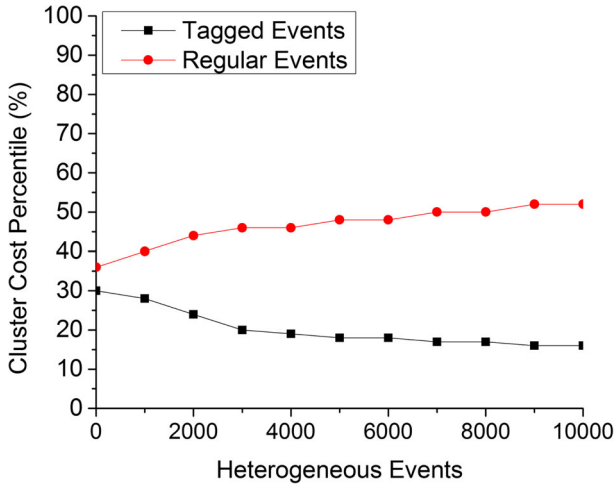

Fig. 11 Heterogeneous Event Processing on Spark Cluster

Acknowledgments
This paper was produced with support from the Ministry of Science and ICT's Broadcasting and Communication Development Fund and may differ from the official opinion of the Ministry of Science and ICT.

About the authors

JAHWAN KOO received the B.S. and M.S. degrees in information engineering from Sungkyunkwan University (SKKU), South Korea, in 1994 and 1996, respectively. He also received the Ph.D. degree in information communication engineering at the School of Information and Communication Engineering, SKKU, in 2005. For more than five years, he was a system engineer and infrastructure architect at Korea Information Systems and LG CNS Co., Ltd. (initially LG-EDS), Seoul, Korea. He joined the School of Information and Communication Engineering, SKKU, as a Research Professor in 2006. He received Post-Doctoral Fellowship in the Computer Science Department, University of Wisconsin-Madison, USA, from 2007 to 2010. He worked for CHAHOO as a Director, from 2010 to 2012 and for Geasoft Co., Ltd as a Head of Research Center, from 2015 to 2016. He joined Consumer & Family Sciences, SKKU, as a Research Professor, from 2016 to 2019. He is currently a Visiting Professor in the College of Software, SKKU. His research interests include big data platform, data mining, natural language processing, computer networking, and cloud computing.

NAWAB MUHAMMAD FASEEH QURESHI is an Assistant Professor at Sungkyunkwan University, Seoul, South Korea. He received Ph. D. in Computer Engineering from Sungkyunkwan University, South Korea, through SAMSUNG scholarship. He was awarded the 1st Superior Research Award from the College of Information and Communication Engineering on account of his research contributions and performance during studies. Currently, he is serving 4 guest editorials as, Lead Guest Editor of Multimedia Systems, Springer. "Role of Deep Learning Models & Analytics in an Industrial Multimedia Environment", Guest Editor of Computers, Materials & Continua Special Issue "Artificial Intelligence and Big Data in Entrepreneurship", Guest Editor of Future Internet Journal Special Issue "Special Issue on Cyber Physical Systems: Prospects, Challenges, and Role in Software Defined Networking and Blockchain" and Guest Editor of Internet Technology Letters "Special Issue on Deep Learning for Future Smart Cities". Also he has served as General Chair Workshop NexGenRAN (Open-RAN: Open Road to Next Generation Mobile Networks) in IEEE Wireless Communications and Networking Conference (WCNC2020) 25th May, Seoul, South Korea and serving as Proceedings Chair in Global Conference on Wireless & Optical Technologies 2020 (GCWOT'20) and General Chair Workshop Open-RAN: Open Road to Next Generation Mobile Networks in IEEE Globecom 2020, Taiwan. He is a reviewer of various prestigious journals such as Future Generation Computer Systems, Transactions on Emerging Telecommunications Technologies (ETT), Wireless Personal Communications (Springer), KSII Transactions on Internet and Information Systems, Journal of Supercomputing (Springer), IEEE Communications Magazine, IEEE Transactions on Industrial Informatics, IEEE Transactions on Industrial Electronics, IEEE Transactions on Industry Applications, IEEE Access, Mathematical Problems in Engineering (Hindawi Publishers), MDPI series of Journals including Symmetry, Electronics, Applied Sciences, Information, and Energies, Journal of Real-Time Image Processing (Springer). He has been a reviewer of various top-tier conferences such as IEEE Globecom2018, IEEE PIMRC 2017, IEEE ICACT 2019, AIIPCC2020, and IEEE ICACT 2020. He has been a TCP in IWWCN2017, CSA2017, IMTIC18, and WCSN2017 conferences and performed as session chairs with ICGCET Denmark, RTCSE19 USA, ICACT 2019 South Korea and RTCSE 2020. He has evaluated several theses as external Ph.D. thesis evaluators. He has facilitated several institutes with Webinars on Big data analysis and Modern Technology convergence and served sessions with keynote talks on convergence with modern technologies. He is an active Senior Member of IEEE, ACM, KSII (Korean Society for Internet Information), and IEICE (Institute of Electronics, Information and Communication Engineers). His research interests include big data analytics, context-aware data processing of the Internet of Things, and cloud computing.

ISMA FARAH SIDDIQUI is an Associate Professor in the Department of Software Engineering, Mehran University of Engineering and Technology, Pakistan. She received her Ph. D. degree in Computer Engineering with distinction from Hanyang University, ERICA, South Korea with the financial support of Higher Education Commission, Pakistan. She received "Best Ph.D. Graduate" Award from the college of Computing, ERICA, Hanyang University. She is reviewer of various renowned SCIE journals such as Wireless Personal Communications, IEEE Access, Future Generation Computer Systems, KSII Transactions on Internet and Information Systems and Journal of Supercomputing. She conducted various technical workshops and technical talks including IMTIC 2018 and 16th Annual symposium at MMC, Pakistan. She's been TPC of various national and international conferences including IMTIC'18 and FIIT'18. Her research interests include Smart Environment, Semantic Web, IoT and Big Data.

ASAD ABBAS is currently serving as Assistant Professor at Faculty of Information Technology, at University of Central Punjab Lahore, Pakistan. He has served as Assistant Professor at Department of Software Engineering at University of Lahore, Lahore Pakistan. He received his BS (Information Technology) degree from "University of the Punjab" Pakistan in 2011. He Joined MS-leading to PhD program in Department of Computer Science and Engineering in Hanyang University ERICA campus at Ansan, South Korea, funded by Higher Education Commission of Pakistan in 2014. He received his Ph.D. degree in Computer Science and Engineering from Hanyang University South Korea in August 2018. His research interests include Software Product Line, Software Requirement Traceability and IoT Applications.

ALI KASHIF BASHIR is a Senior Lecturer/Associate Professor and Course Leader of BSc (H) Computer Forensics and Security at the Department of Computing and Mathematics, Manchester Metropolitan University, United Kingdom. He is also holding Adjunct Professor Position at National University

of Science and Technology, Pakistan. He is a senior member of IEEE, invited member of IEEE Industrial Electronic Society, member of ACM, and Distinguished Speaker of ACM. His past assignments include HYPERLINK "https://www.setur.fo/fo/setrid/tidindi/nyggjur-lektari/" Associate Professor of ICT, University of the Faroe Islands, Denmark; HYPERLINK "http://www.ist.osaka-u.ac.jp/english/index.html" Osaka University, Japan; HYPERLINK "https://www.nara-k.ac.jp/english/" Nara National College of Technology, Japan; the HYPERLINK "https://www.nfri.re.kr/eng/index" National Fusion Research Institute, South Korea; HYPERLINK "https://www.kospo.co.kr/english/" Southern Power Company Ltd., South Korea, and the HYPERLINK "http://english.seoul.go.kr/" Seoul Metropolitan Government, South Korea. He has worked on several research and industrial projects of South Korean, Japanese and European agencies and Government Ministries. He received his Ph.D. in computer science and engineering from HYPERLINK "https://www.korea.edu/" Korea University South Korea. He has authored over 150 research articles; received funding as PI and Co-PI from research bodies of South Korea, Japan, EU, UK and Middle East; supervising/co-supervising several graduate (MS and PhD) students. His research interests include internet of things, wireless networks, distributed systems, network/cyber security, network function virtualization, machine learning, etc. He is serving as the Editor-in-chief of the HYPERLINK "https://cmte.ieee.org/futuredirections/tech-policy-ethics/" IEEE FUTURE DIRECTIONS NEWSLETTER. He is also serving as area editor of KSII Transactions on Internet and Information Systems; associate editor of IEEE Access, IET Quantum Computing. He is leading many conferences as a chair (program, publicity, and track) and had organized workshops in flagship conferences like IEEE Infocom, IEEE Globecom, IEEE Mobicom, etc.

Declaration
I would like to thank the editorial desk for offering this opportunity to express the declarations requested by the editorial desk. Please consider following sub-title declarations as part of the submission process.

Authors' contributions
The authors have contributed in such a manner as: 1st author: He has performed major portions of experiments and have written the manuscript 2nd author: He is responsible to answer the corresponding author queries as well as have performed directed acyclic graph experimentation 3rd author: She is responsible for debugging evaluations in the simulations. 4th author: He is responsible to evaluate the modeling values extracted by the execution of ecosystem. 5th author: He is managing the editing of manuscript in terms of technical as well as written English.

Funding
This paper was produced with support from the Ministry of Science and ICT's Broadcasting and Communication Development Fund and may differ from the official opinion of the Ministry of Science and ICT.

Availability of data and materials
The data related to the manuscript is available with the authors and could be produced if required.

Competing interests
There is not any conflict of interest among the authors related to the content disclosed at the disposal of manuscript and the authors have a mutual consent on all concerned points related to the manuscript.

Author details
[1]College of Software, Sungkyunkwan University, Seoul, South Korea. [2]Department of Computer Education, Sungkyunkwan University, Seoul, South Korea. [3]Department of Software Engineering, Mehran University of Engineering & Technology, Jamshoro, Pakistan. [4]Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan. [5]Department of Computing and Mathematics, Manchester Metropolitan University,Manchester, UK.

Received: 21 January 2020 Accepted: 10 August 2020
Published online: 14 September 2020

References

1. Gaber M, Zaslavsky A, Krishnaswamy S (2005) Mining data streams: a review. ACM Sigmod Rec 34(2):18–26
2. Denning PJ (1990) The science of computing: Saving all the bits. American Sci 78(5):402–405
3. Vega M, Perra C, De Turck F, Liotta A (2018) A review of predictive quality of experience management in video streaming services. IEEE Trans Broadcast 64(2):432–445
4. de Assuncao M, da Silva Veith A, Buyya R (2018) Distributed data stream processing and edge computing: A survey on resource elasticity and future directions. J Netw Comput Appl 103:1–17
5. Krempl G, Žliobaite I, Brzeziński D, Hüllermeier E, Last M, Lemaire V, Noack T, Shaker A, Sievi S, Spiliopoulou M, et al. (2014) Open challenges for data stream mining research. ACM SIGKDD explor newsl 16(1):1–10
6. Wu X, Zhu X, Wu G-Q, Ding W (2013) Data mining with big data. IEEE Trans Knowl Data Eng 26(1):97–107
7. Streaming SQL Analytics for Kafka & Kinesis. https://sqlstream.com. Accessed 11 Dec 2019
8. Software Inc. T Global Leader in Integration and Analytics Software. https://www.tibco.com/. Accessed 11 Dec 2019
9. Inc. I Computer hardware company. http://www.ibm.com. Accessed 11 Dec 2019
10. striim stream with two i's for integration and intelligence. https://www.striim.com/. Accessed 11 Dec 2019
11. Apache Org Welcome to The Apache Software Foundation!. https://www.apache.org/. Accessed 11 Dec 2019
12. Hoffman S (2013) Apache Flume: Distributed Log Collection for Hadoop. Packt Publishing Ltd, USA
13. Zaharia M, Xin R, Wendell P, Das T, Armbrust M, Dave A, Meng X, Rosen J, Venkataraman S, Franklin M, et al. (2016) Apache spark: a unified engine for big data processing. Commun ACM 59(11):56–65
14. Jain A, Nalya A (2014) Learning Storm. Packt Publishing Ltd, USA
15. Apache nifi Welcome to The Apache Nifi. http://nifi.apache.org. Accessed 11 Dec 2019
16. Apache Apex Welcome to The Apache Apex
17. Apache Kafka Welcome to The Apache Kafka. http://kafka.apache.org. Accessed 11 Dec 2019
18. Apache Samza Welcome to The Apache Samza. samza.apache.org. Accessed 11 Dec 2019
19. Apache Flink Welcome to The Apache Flink. http://flink.apache.org. Accessed 11 Dec 2019
20. Apache Beam Welcome to The Apache Beam. http://beam.apache.org. Accessed 11 Dec 2019
21. Apache Ignite Welcome to The Apache Ignite. http://ignite.apache.org. Accessed 11 Dec 2019
22. Tatbul N (2010) Streaming data integration: Challenges and opportunities. In: 2010 IEEE 26th International Conference on Data Engineering Workshops (ICDEW 2010). IEEE, USA. pp 155–158
23. Watanabe Y, Yamada S, Kitagawa H, Amagasa T (2007) Integrating a stream processing engine and databases for persistent streaming data management. In: International Conference on Database and Expert Systems Applications. Springer, USA. pp 414–423
24. Atzori L, Iera A, Morabito G (2010) The internet of things: A survey. Comput Netw 54(15):2787–2805
25. Vural S, Navaratnam P, Wang N, Wang C, Dong L, Tafazolli R (2014) In-network caching of internet-of-things data. In: 2014 IEEE International Conference on Communications (ICC). IEEE, USA. pp 3185–3190
26. Stonebraker M, Çetintemel U, Zdonik S (2005) The 8 requirements of real-time stream processing. ACM Sigmod Rec 34(4):42–47
27. Gaur P, Tahiliani M (2015) Operating systems for iot devices: A critical survey. In: 2015 IEEE Region 10 Symposium. pp 33–36. IEEE
28. Kang Y-S, Park I-H, Rhee J, Lee Y-H (2015) Mongodb-based repository design for iot-generated rfid/sensor big data. IEEE Sensors J 16(2):485–497
29. Ranjan R (2014) Streaming big data processing in datacenter clouds. IEEE Cloud Comput 1(1):78–83
30. Kamburugamuve S, Fox G, Leake D, Qiu J (2013) Survey of distributed stream processing for large stream sources. Grids Ucs Indiana Edu 2:1–16
31. Lemon J, Wang Z, Yang Z, Cao P (2004) Stream engine: A new kernel interface for high-performance internet streaming servers. In: Web Content Caching and Distribution. Springer, USA. pp 159–170
32. Liew C, Atkinson M, van Hemert J, Han L (2010) Towards optimising distributed data streaming graphs using parallel streams. In: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing. ACM, USA. pp 725–736

33. Saad A, Park S (2019) Decentralized directed acyclic graph based dlt network. In: Proceedings of the International Conference on Omni-Layer Intelligent Systems. ACM, USA. pp 158–163
34. Qureshi N, Bashir A, Siddiqui I, Abbas A, Choi K, Shin D (2018) A knowledge-based path optimization technique for cognitive nodes in smart grid. In: 2018 IEEE Global Communications Conference (GLOBECOM). IEEE, USA. pp 1–6
35. Qureshi N, Siddiqui I, Unar M, Uqaili M, Nam C, Shin D, Kim J, Bashir A, Abbas A (2019) An aggregate mapreduce data block placement strategy for wireless iot edge nodes in smart grid. Wirel Pers Commun 106(4):2225–2236
36. Siddiqui I, Qureshi N, Shaikh M, Chowdhry B, Abbas A, Bashir A, Lee S-J (2019) Stuck-at fault analytics of iot devices using knowledge-based data processing strategy in smart grid. Wirel Pers Commun 106(4):1969–1983
37. Zhu F, Wu W, Zhang Y, Chen X (2019) Privacy-preserving authentication for general directed graphs in industrial IoT. Inf Sci 502:218–228
38. Kotilevets I, Ivanova I, Romanov I, Magomedov S, Nikonov V, Pavelev S (2018) Implementation of directed acyclic graph in blockchain network to improve security and speed of transactions. IFAC-PapersOnLine 51(30):693–696
39. Deng C, Yang E, Liu T, Tao D (2020) Two-Stream Deep Hashing With Class-Specific Centers for Supervised Image Search. IEEE Trans Neural Netw Learn Syst 31:2189–2201. https://doi.org/10.1109/TNNLS.2019.2929068
40. Saw J, Yang M, Mo T (1984) Chebyshev inequality with estimated mean and variance. Am Stat 38(2):130–132
41. Duda P, Jaworski M, Pietruczuk L, Rutkowski L (2014) A novel application of hoeffding's inequality to decision trees construction for data streams. In: 2014 International Joint Conference on Neural Networks (IJCNN). IEEE, USA. pp 3324–3330
42. Qureshi N, Siddiqui I, Abbas A, Bashir A, Choi K, Kim J, Shin D (2019) Dynamic container-based resource management framework of spark ecosystem. In: 2019 21st International Conference on Advanced Communication Technology (ICACT). IEEE, USA. pp 522–526
43. Siddiqui IF, Qureshi NMF, Chowdhry BS, Uqaili MA (2020) Pseudo-Cache-Based IoT Small Files Management Framework in HDFS Cluster. Wirel Personal Commun
44. Qureshi NMF, Siddiqui IF, Abbas A, Bashir AK, Nam CS, Chowdhry BS, Uqaili MA (2020) Stream-Based Authentication Strategy Using IoT Sensor Data in Multi-homing Sub-aqueous Big Data Network. Wirel Personal Commun.1–13
45. Siddiqui IF, Qureshi NMF, Chowdhry BS, Uqaili MA (2019) Edge-node-aware adaptive data processing framework for smart grid. Wirel Personal Commun 106(1):179–189
46. Qureshi NMF, Shin DR, Siddiqui IF, Chowdhry BS (1374) Storage-tag-aware scheduler for hadoop cluster. IEEE Access 5:2–13755
47. Qureshi NMF, Shin DR (2016) RDP: A storage-tier-aware Robust Data Placement strategy for Hadoop in a Cloud-based Heterogeneous Environment. TIIS 10(9):4063–4086

Publisher's Note